DevOps Exam Assignment

May 2023

In this exam assignment you will design and implement an infrastructure and development environment for a web application that enables users to search for contact information of teachers using their 'short name'.

The Vue web application connects to a backend written in Node. The contact information is stored in this backend application. The Node application uses an SQLite database to store the contacts.

Your task will be to run and test this application. Additionally, you will have to set up the continuous integration pipeline for this application and containerize the application.

Entry Requirements

- It is mandatory to hand in all the *practice assignments* and have them approved by your teacher, before your final assignment can be handed in. If you haven't done so, please contact your teacher ASAP.
- For each requirement (see below) you should explain how you solved the problem and why you solved it in this way. This should be described in your own words in a way that is understandable for the teacher that is grading your work. You can either write it down in a Markdown file included in your repository, or in comments in the appropriate source file (in the latter case, please include references to the locations of these comments in the Markdown file).
- Make sure you use a repository in the Saxion Gitlab organization, created via https://repo.hboictlab.nl
- Upload a zipped export of your Gitlab project to Blackboard (Settings → General → Advanced → Export Project).
  Please mention the URL of your repository in the comment box on Blackboard as well.

Exam Rules

- The deadline for handing in the assignment is Monday 5th of June 9:00 (week 4.5).
- The assignment is done in exam groups of 1 or 2 students that are registered on Blackboard.
- Your grade is determined based on the oral exam that will take place after you handed in your assignment.
- You will only receive points for elements of your work that you are able to explain during the oral exam. For example: if your code works but you cannot explain what it does or why it works like that, it won't count towards your final grade.
- It is **not** allowed to receive assistance from anyone (human or AI) outside of your exam group, except for your teacher. It is also **not** allowed to give assistance to anyone else.
- You *are* allowed to use online posts, articles, tutorials, books, videos. You must add references[1] for all sources and code fragments that you used in your text/code.

[1] https://libguides.murdoch.edu.au/IEEE

Functional Requirements

1. Add data to the backend

In the template folder you will find a folder containing a backend application. This is a Node application provided by us that returns data in JSON format. If you run the application (using NPM) and navigate to http://localhost:5000/status you will get a list of contacts (which should be empty at this point):

{"host":"localhost","message":"Successfully connected to the backend.", "contacts":[]}

In order to add contacts to the backend you should create a POST request with the proper format (see `add_contacts.sh`). You can use `curl` to create such a POST request. Make sure you set the content type to `application/json` in your Curl command.

In the folder '*seed-application*' you will find a Bash script called `add_contacts.sh`. Edit this script so that it sends a list of contacts to the backend. The list should be read from the `data.txt` file.

2. Containerize backend and frontend applications

The backend and frontend application should be containerized using Docker and Docker Compose. There should be separate backend and frontend containers. The frontend should be served using Nginx.

You should facilitate a convenient way to build, run and stop the containers by writing a script. Make sure that intermediate files created by running your application outside of Docker are not included in the image.

3. Containerization of an external database and seed script

Extend your compose file so that the data is not stored in SQLite in the backend but in an external PostgreSQL database (which is run in a container). For this to work you should set the following environment variables in the backend and the PostgreSQL container:

```
POSTGRES_USER
POSTGRES_PASSWORD
POSTGRES_DB
```

In the backend container you should also set the following environment variables:

- `NODE_ENV` (set its value to 'production' so that Node knows that it needs to run in production mode)
- `POSTGRES_HOST` (set the name of the server which the backend needs to connect to)
- `POSTGRES_PORT` (set the port of the server which the backend needs to connect to)

Optionally, you could create an additional container for filling the database with the script that you have written in step 1. Make sure this script is only executed when the database is empty.

4. Continuous Integration

Set up continuous integration for the app. The continuous integration should consist of at least two stages. In the first stage you have to lint the backend application. In the second stage you need to run the provided (and supplemented by you) unit tests. You might have to fix some linting issues first before the linting is successful!

Make sure that the pipeline is automatically executed when changes are pushed to the main branch or to a merge request. Both the linting and the unit testing will create a report. Make sure to store these reports as artifacts in Gitlab (with an expiration date of one month).

Please note: to run the unit tests, you should first start the backend server in *testing* mode (`NODE_ENV`) and then run the unit tests.

5. Create an infrastructure to run the application

Create an appropriate infrastructure for the application in AWS using Terraform. The infrastructure does not need to be high availability (yet). If you do not manage to do so using Terraform you are allowed to manually create the infrastructure (this will lose you points of course). Your Terraform configuration should be placed in the provided `infra` folder.

The backend should be installed on an EC2 instance (manually for now), the frontend may be served as a static website.

6. Automate the application deployment

Automate the deployment of the backend, so that when we commit to the main branch the Gitlab CI:

- creates the necessary Docker image(s)
- uploads the image(s) to the private container registry for your Gitlab repository
- connects to the EC2 instance using SSH, pulls the Docker image(s) and runs the image(s).

To successfully achieve this, you should copy the contents of your AWS private SSH key (that you use to connect to the instance) to an environment variable in Gitlab CI (Settings -> CI/CD -> Variables, click on "Expand"). In your `.gitlab-ci.yml` file you can use this variable to connect to the EC2 instance using SSH.

When using SSH the client will check whether the server is a known server. For our automated deployment this is not very useful (as it would require someone to type in 'yes'). To omit this check, you can pass the `-o StrictHostKeyChecking=no` option to SSH.

7. Create a high availability backend

Extend your Terraform configuration with a high availability backend. You do not need to make the *database* highly available (i.e. redundant), one instance suffices (but of course in real life you would need to think about this).

Make sure the instances are not accessible directly from outside of your network. Remember to update the frontend to connect to your updated infrastructure. To validate whether your infrastructure is set up correctly, check the 'Hostname' information logged to your browser's console in the frontend. You should see different hostnames while refreshing the page.

Process Requirements

While doing the assignment, you should follow a proper software development process. This includes things like tracking tasks using Gitlab issues, using Git for collaborating on the code, and using branches and merge requests to facilitate code reviews. This should all be evident from your Gitlab repository.